Accessing data from the web
Crawling, scraping, and APIs

Snorre Ralund

Todays message

- Utilize the datasources around you. (Job data and crime)
- Knowing how to create your own custom datasets pulling information from many different sources.
- You should know all the tricks, but use them with care.

Agenda

- The basics of webscraping
  - Connecting, Crawling, Parsing, Storing, Logging.
- Hacks: Backdoors, url construction, and analysis of a webpage.
- Reliability of your datacollection.
- Screen-scraping - Automated browsing
  - Interactions:
    - Login in, scrolling, pressing buttons.
- APIs
  - Authentication
  - Building queries

Ethics / Legal Issues

- If a regular user can’t access it, we shouldn’t try to get it (That is considered hacking)https://www.dr.dk/nyheder/penge/gjorde-opmaerksom-paa-cpr-hul-nu-bliver-han-politianmeldt-hacking (https://www.dr.dk/nyheder/penge/gjorde-opmaerksom-paa-cpr-hul-nu-bliver-han-politianmeldt-hacking).
- Don't hit it to fast: Essentially a DENIAL OF SERVICE attack (DOS). Again considered hacking (https://www.dr.dk/nyheder/indland/folketingets-hjemmeside-ramt-af-hacker-angreb).
- Add headers stating your name and email with your requests to ensure transparency.
- Be careful with copyrighted material.
- Fair use (don't take everything)
- If monetizing on the data, be careful not to be in direct competition with whom you are taking the data from.

11. DEC. 2015 KL. 13.13

Folketingets hjemmeside ramt af hacker-angreb

Forsøg på at komme ind på Folketingets hjemmeside resulterer i besked om, at siden ikke er tilgængelig.

Folketinget er blevet ramt af et hacker-angreb, bekræfter Finn Tørngren Sørensen, presseansvarlig i Folketinget, over for Avisen.dk.

Siden fredag formiddag har man fået beskeden "Denne webside er ikke tilgængelig", hvis man har forsøgt at komme ind på Folketingets hjemmeside, ft.dk.

- Det er rigtigt, at der er lukket for den eksterne adgang til Folketingets hjemmeside. Vi er under et såkaldt 'Denial of service"-angreb, og det har vi været siden klokken ti i formiddags. Det fungerer på den måde, at vi får så mange opkald til vores hjemmeside, at systemet bliver overbelastet. Derfor har vi måttet lukke ned for adgangen, siger han.

Folketinget har endnu ikke noget overblik over, hvem der står bag hacker-angrebet, eller hvornår hjemmesiden kan komme op at køre igen.

/ritzau/

Setting up the essentials:

Good practices:

- Transparency
- Ratelimiting
- Reliability

In [2]:
```
# Transparent scraping
import requests
session = requests.session()
#session.headers['email'] = 'youremail'
#session.headers['name'] = 'name'
session.headers
```

Out[2]:
```
{'User-Agent': 'python-requests/2.18.4', 'Accept-Encoding': 'gzip, deflate', '
Accept': '*/*', 'Connection': 'keep-alive'}
```

A quick tip is that you can change the user agent to a cellphone to obtain more simple formatting of the html.

In [3]:
```
# Control the pace of your calls
import time
def ratelimit():
    "A function that handles the rate of your calls."
    time.sleep(1)
# Reliable requests
def request(url,iterations=10,exceptions=(Exception)):
    """This module ensures that your script does not crash from connection errors.
        iterations : Define number of iterations before giving up.
        exceptions: Define which exceptions you accept, default is all.
    """
    for iteration in range(iterations):
        try:
            # add ratelimit function call here
            ratelimit() # !!
            response = session.get(url)
            return response # if succesful it will end the iterations here
        except exceptions as e: #  find exceptions in the request library requests
.exceptions
            print(e) # print or log the exception message.
    return None # your code will purposely crash if you don't create a check funct
ion later.
```

In [76]:
```
# Interactive browsing
from selenium import webdriver
path2gecko = '/Users/axelengbergpallesen/Downloads/geckodriver' # define path to y
our geckodriver
browser = webdriver.Firefox(executable_path=path2gecko)
browser.get('https://www.facebook.com')
```

Now we are ready to get some data

Collecting data from the web: a quick example

So let's get some data.

Lets say we wanted to obtain high frequency and geolocated information about the danish jobmarket. Then we might build a scraper for jobfinder.dk (https://www.jobfinder.dk/), jobindex.dk (https://www.jobindex.dk/) or jobnet.dk (https://job.jobnet.dk/CV/FindWork/).

In [105]:
```
url = 'https://www.jobfinder.dk/jobs/category/it-konsulent'
base = 'https://www.jobfinder.dk/'
response = request(url)
```

In [23]:
```
#print(response.text) # inspect the response to see if it is what you expected.
```

In [106]:
```
html = response.text
link_nodes = html.split('user-jobs__content-item')[1:]
for node in link_nodes[0:5]:
    link_node = node.split('h2')[1]
    link = link_node.split('href="')[1]
    print(link.split('"')[0])
```

```
/job/senior-bi-konsulent-og-arkitekt-30080
/job/vi-soeger-skarpe-konsulenter-40316
/job/ambitioese-juniorkonsulenter-33672
/job/make-impact-it-consultant-33146
/job/sikkerhedskonsulent-til-succesfuld-it-virksomhed-41132
```

Loop over the links and dump the data

In [ ]:
```
# make directory
# ! mkdir scraping_examples
```

In [74]:
```
# define path
base_path = 'scraping_examples/'
filename = base_path+'jobfinder_data'
f = open(filename,'w')
import json
# make loop
html = response.text
link_nodes = html.split('user-jobs__content-item')[1:]
for node in link_nodes[0:5]:
    link_node = node.split('h2')[1]
    link = link_node.split('href="')[1]
    new_url = base+link
    response = request(new_url)
    data = {new_url:response.text}

    f.write(json.dumps(data))
    f.write('\n\r')
f.close()
```

Exercise 1

Get links to the articles listed on this link (https://www.dr.dk/search/Result?filter_facet_universe=Nyheder&query=kv17&sort=Newest): https://www.dr.dk/search/Result?filter_facet_universe=Nyheder&query=kv17&sort=Newest (https://www.dr.dk/search/Result?filter_facet_universe=Nyheder&query=kv17&sort=Newest)

Loop over the links and save them to a file.

Paging extra

- Use selenium to click on more results or figure out the paging mechanism looking at the activity in the network monitoring of your browser.

Reliability Extra

- When writing the file: provide information about servertime (import the time module), the link, other information.

Tricks: Backdoors, pseudo-apis, and url construction.

Static or dynamic html pages

You browser will either get a set of instructions (in javascript) on how to build a page and how interactions will change it, or it will receive a pre-compiled html document.

Lets inspect another jobsite: jobnet.dk (https://job.jobnet.dk/CV/FindWork/).

In [ ]:
```
# Another examples
url = 'https://job.jobnet.dk/CV/FindWork/' # define the link
# get the raw html
response = request(url)
html = response.text
```

In [ ]:
```
html # inspect to see if it matches
```

This looks a lot shorter. This points us to the fact that the page is build dynamically.

Now we can search for instructions and the data that it uses to construct the front.

In [86]:
```
# get data through the backdoor
# backdoor link
url = 'https://job.jobnet.dk/CV/FindWork/Search?offset=20'
response = request(url)
```

In [88]:
```
#response.json()
```

Constructions of urls

A nice trick is to understand how urls are constructed to communicate with a server.

Lets look at how jobindex.dk (https://www.jobindex.dk/) does it.

This can help you navigate the page, without having to parse information from the html or click any buttons.

- / is like folders on your computer.
- ? entails parameters
- = defines a variable: e.g. pageid=1000 or offset = 100 or showNumber=20
- & separates different parameters.
-
  - is html for whitespace

Exercise 2

Find the backdoor to the polling data behind the visualization at b.dk (https://www.b.dk/politiko/barometeret (https://www.b.dk/politiko/barometeret)).

- Follow the above link and look at the network activity for data related responses: .csv xml json
- Download the data and parse it using xml2dict (conda install -c conda-forge xmltodict):

```
data = xmltodict.parse(text)
```

extra

Find links to all historical data (either by analyzing the path or by locating the masterfile in the network activity). And set up a loop collecting them all.

extra 2

Download precompiled polling data from github:

- ***link to a compilation of danish polling data (https://github.com/erikgahner/polls)***

In [ ]:
```
import xmltodict
#xmltodict.parse(response.text)
```

Navigation and Parsing

HTML Scraping

Web Scraping: Automated downloading of web pages and the extraction of specific information from it.

Snowball crawling: Finding links, following links, find links, follow links.

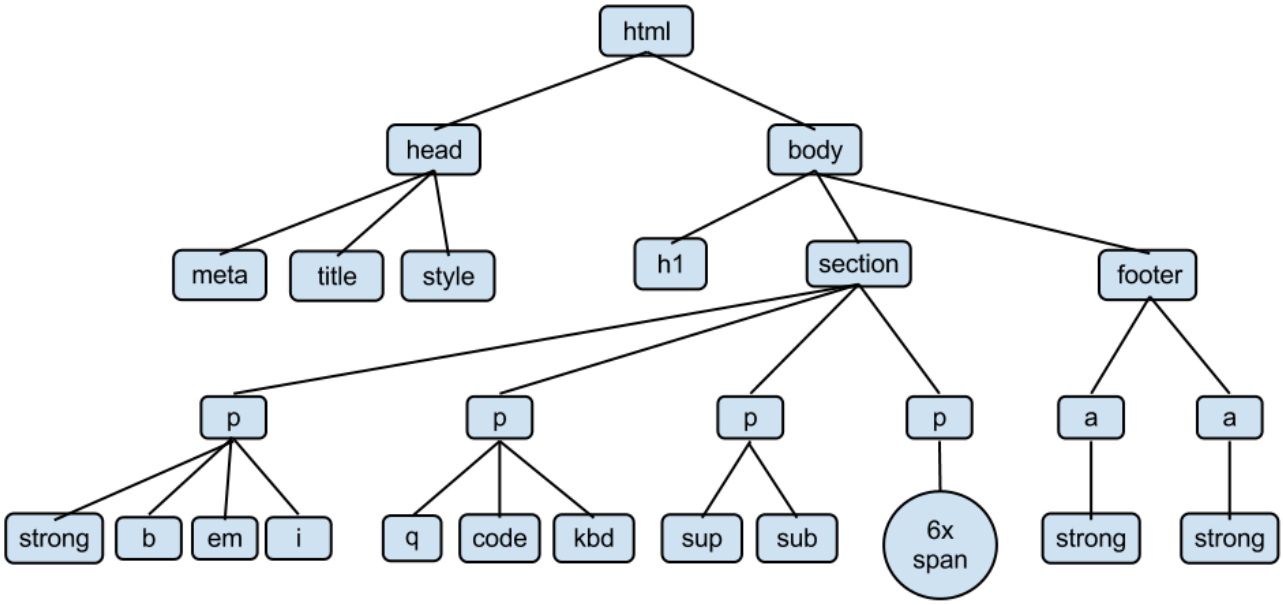
Parsing and extracting information

Retrieving target information from unstructured text.

Parsing is used both while crawling and navigating the domain you are scraping from

HTML data

- Tree structure.
  - Children, siblings, parents - descendants.
    - Ids and attributes



How do we find our way around this tree?

- selectors
- splitting and regex
- traversing html trees (beautifulsoup)

Let's download a module that can parse this tree for us.

```
conda install -c asmeurer beautiful-soup
```

or `pip install beautifulsoup4`

Selectors

- quick but has to be hardcoded and therefore more likely to break

In [ ]:
```
# selenium example
browser.get('https://www.facebook.com')
# find login button.
```

Splitting and searching with Regex

Regex is a general language for searching strings.

Extremely useful for all extractign information from all sorts of unstructured text data.

... however you have to learn it.

Custom parsing using Regex

Won't go into the details here, but working with text this will come in very handy. Play around with it here (http://regexr.com/) or in this notebook.

- + = 1 or more times
- * = 0 or more times
- {3} = exactly three times
- ? = once or none
- \ = escape character, used to find characters that has special meaning with regex: e.g. + *
- () = within the paranthesis is the pattern I care about.
- [] = set of characters
- ^ = applied within a set, it becomes the inverse of the set defined.
- . = any characters except line break
- | = or statement. p|b find characters a or b.

In [186]:
```
import re
s = 'abcdefg123456789'
re.split('bc|56',s)# splitting by a complex pattern
```

Out[186]:
```
['a', 'defg1234', '789']
```

Advanced parsing using Beautifulsoup

Beautifulsoup is a powerful parser build on regex, transforming the raw html into traversable tree of objects.

Here I want to show you how to use it for grabbing data about the KV17 election: https://www.altinget.dk/kandidater/kv17/stemmeseddel.aspx (https://www.altinget.dk/kandidater/kv17/stemmeseddel.aspx)

Let us analyze the page.

In [89]:
```
url = 'https://www.altinget.dk/kandidater/kv17/stemmeseddel.aspx?kommune=13'
response = request(url)
#for i in range(1,86):
#    url = 'www.altinget.dk/kandidater/kv17/stemmeseddel.aspx?kommune=%d'%i
```

In [98]:
```
html = response.text
from bs4 import BeautifulSoup
soup = BeautifulSoup(html,'lxml')
```

In [107]:
```
node = soup.find('dl',{'class':'candidates-list-parties'}) # find, findall
```

In [103]:
```
# children, siblings, parents.
```

In [132]:

```
candidates = []
for child in list(node.children):
    if child['class'][0] == "candidates-party-name":
        party = child.text
        continue
    name = child.a.text
    link = child.a['href']
#    print(name,link)
    row = [party,name,link]
    candidates.append(row)
```

In [134]:
```
import pandas as pd
df = pd.DataFrame(candidates,columns = ['party','name','link'])
df.head()
```

Out[134]:

	party	name	link
0	02. Christiania-Listen	Andreas Bennetzen	/kandidater/kv17/andreas-bennetzen
1	02. Christiania-Listen	Britta Lillesøe	/kandidater/kv17/Britta-Lillesoee
2	02. Christiania-Listen	Klaus Haase	/kandidater/kv17/Klaus-Haase
3	02. Christiania-Listen	Lis Brandstrup	/kandidater/kv17/Lis-Brandstrup
4	02. Christiania-Listen	Carl Oskar Strange	/kandidater/kv17/carl-oskar-strange

In [137]:
```
# Getting information about each candidate

url = 'https://www.altinget.dk/kandidater/kv17/3010-frank-jensen'
response = request(url)
html = response.text
soup = BeautifulSoup(html,'lxml')
```

In [142]:
```
meta_nodes = soup.findAll('h3',{'class':'subsection-title h4'})
```

In [143]:
```
for meta_node in meta_nodes:
    if meta_node.text == 'Fakta':
        break
```

In [145]:
```
# search for siblings.
```

In [148]:
```
node = meta_node.find_next('dl')
node
```

Out[148]:
```
<dl>
<dt>Valgt i</dt><dd>2009, 2013</dd>
<dt>Uddannelse</dt><dd>Kandidatuddannelse</dd>
<dt>Beskæftigelse</dt><dd>Borgmester</dd>
<dt>Alder</dt><dd>56 år</dd>
</dl>
```

In [166]:
```
candidate_data = {}
for cat in node.findAll('dt'):
    category_title = cat.get_text()
    category_point = cat.find_next_sibling().get_text()
    print(category_title,category_point)
    candidate_data[category_title] = category_point
```

```
Valgt i 2009, 2013
Uddannelse Kandidatuddannelse
Beskæftigelse Borgmester
Alder 56 år
```

In [167]:
```
answers = soup.findAll('h3',{'class':'panel-title'})
```

In [168]:
```
for answer in answers:
    title = answer.text
    candidate_answer = answer.find_next('div',{'class':'answer-candidate tooltip i
n top'}).previous
    candidate_data[title] = candidate_answer
```

In [169]:
```
#candidate_data
```

Out[169]:

```
{'Alder': '56 år',
 'BESKÆFTIGELSE': 'Delvist enig',
 'Beskæftigelse': 'Borgmester',
 'BÆREDYGTIGHED': 'Delvist enig',
 'DAGINSTITUTIONER': 'Delvist enig',
 'ENERGI': 'Helt enig',
 'ERHVERV': 'Delvist uenig',
 'FLYGTNINGE': 'Delvist uenig',
 'FOLKESKOLE': 'Delvist uenig',
 'FOLKESUNDHED': 'Hverken/eller',
 'FOREBYGGELSE': 'Delvist enig',
 'FRIVILLIGHED': 'Delvist uenig',
 'IDRÆT': 'Helt enig',
 'INTEGRATION': 'Delvist uenig',
 'KLIMA': 'Helt enig',
 'KOMMUNESKAT': 'Delvist uenig',
 'KULTUR': 'Helt uenig',
 'MILJØ': 'Helt enig',
 'TRANSPORT': 'Helt enig',
 'Uddannelse': 'Kandidatuddannelse',
 'Valgt i': '2009, 2013',
 'ÆLDRE': 'Helt enig',
 'ÆLDREPLEJE': 'Delvist uenig',
 'ØKONOMI': 'Delvist uenig'}
```

Hvem er den vageste politiker?

Exercise 3: Find the facebook ids of all candidates in the KV17 election

- Loop through the first 5 muncipalities and collect all candidates.
- Get the facebook link from each candidate's page.
- Parse all social media links in the section (Mere information), without knowing the classes in advance.

extra: Parse all information on the page including answers to the candidate test, and count how many 'delvist' answers each politician has. Who is the vaguest politician in denmark?

extra: search for the candidate and kv17 on dr.dk (https://www.dr.dk/search/Result?filter_facet_universe=Nyheder&query=candidate+kv17 (https://www.dr.dk/search/Result?filter_facet_universe=Nyheder&query=candidate+kv17))

In [180]:
```
url = 'https://www.altinget.dk/kandidater/kv17/jacob-bundsgaard'

response = request(url)
```

In [181]:
```
html = response.text
```

In [182]:
```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html,'lxml')
```

In [183]:
```
nodes = soup.findAll('h3',{'class':'subsection-title h4'})
```

In [188]:
```
node = [node for node in nodes if node.text=='Mere information'][0]
node
```

Out[188]:
```
<h3 class="subsection-title h4">Mere information</h3>
```

In [189]:
```
node = node.find_next('dl')
```

In [190]:
```
for link_node in node.find_all_next('a'):
    break
```

In [191]:
```
#link_node.a['href']
link_node
```

Out[191]:
```
<a href="https://www.instagram.com/jacobbundsgaard/">Instagram </a>
```

In [192]:
```
SoMe = {}
for link_node in node.find_all_next('dd'):
    link_node = link_node.a
    name = link_node.text.strip()
    link = link_node['href']
    SoMe[name] = link
```

In [193]:
```
if 'Facebook' in SoMe:
```

Storing data

Storing processed data, logging activity.

- in csv,
- storing in json,
- as picklefile,
- as dataframe Install a csv reader and writer:

```
conda install -c anaconda unicodecsv
```

or

```
pip install unicodecsv
```

Now let's try parsing more structured data: Tables.

If you are not sure always ask google: "how to parse html table"

Hit the following link (https://www.basketball-reference.com/leagues/NBA_2017.html).

In [124]:
```
import requests
response = requests.get('https://www.basketball-reference.com/leagues/NBA_2017.html') # get the html
```

In [121]:
```
from bs4 import BeautifulSoup # import parser

bsobj = BeautifulSoup(response.text) # parse the response using beautifulsoup
```

In [122]:
```
tables = bsobj.findAll('table') # using the tag to locate tables
table = bsobj.select('#confs_standings_E > thead > tr > th.poptip.sort_default_asc
.left') # using a specific css selector to locate specific table
```

lets look at the tables

In [144]:
```
for table in tables[0:1]: # iterating through the tables
    header = table.findAll('thead')[0] # thead is standard notation for table head
er
    data = table.findAll('tbody')[0] # tbody is standard notation for the data

    head = []
    for row in header.findAll('th'): # collect data from header.
        #print(row['aria-label'])#.attrs)#,row.text)
        head.append(row['aria-label'])
 #   print(head)
    tbody = [] # container for the data
    for row in data.findAll('tr'): # tr is standard notation for table rows.
#         print row
        tempoary_row = []
        for cell in row.children: # think of children as indentation in python.
            print(cell.get_text())
            tempoary_row.append(cell.get_text())
        tbody.append(tempoary_row)
#         break
tbody
```

```
Boston Celtics* (1)
53
29
.646
—
108.0
105.4
2.25
Cleveland Cavaliers* (2)
51
31
.622
2.0
110.3
107.2
2.87
Toronto Raptors* (3)
51
31
.622
2.0
106.9
102.6
3.65
Washington Wizards* (4)
49
33
.598
4.0
109.2
107.4
1.36
Atlanta Hawks* (5)
43
39
.524
10.0
103.2
104.0
-1.23
Milwaukee Bucks* (6)
42
40
.512
11.0
103.6
103.8
-0.45
Indiana Pacers* (7)
```

In [147]:
```
for table in tables[0:1]: # pick only the first table
    header = table.findAll('thead')[0]
    data = table.findAll('tbody')[0]
    head = [] # define container for the header
    for column in header.findAll('th'): # collect data from header.
        head.append(column['aria-label']) # append to header container
    rows = [] # define container for the rows
    for row in data.findAll('tr'):
        temp_row = [] # define tempoary row container
        for cell in row.children: # think of children as indentation in python.
            val = cell.get_text()
            try:
                val = float(val)
            except:
                pass
            temp_row.append(val) # append to tempoary row
        rows.append(temp_row) # append tempoary row to row container

import pandas as pd
df = pd.DataFrame(columns=head,data=rows) # define the dataframe
```

In [148]:
```
df # look at the dataframe
```

Out[148]:

	Eastern Conference	Wins	Losses	Win-Loss Percentage	Games Behind	Points Per Game	Opponent Points Per Game	Si Ra Sy
0	Boston Celtics* (1)	53.0	29.0	0.646	—	108.0	105.4	2.2
1	Cleveland Cavaliers* (2)	51.0	31.0	0.622	2	110.3	107.2	2.8
2	Toronto Raptors* (3)	51.0	31.0	0.622	2	106.9	102.6	3.6
3	Washington Wizards* (4)	49.0	33.0	0.598	4	109.2	107.4	1.3
4	Atlanta Hawks* (5)	43.0	39.0	0.524	10	103.2	104.0	-1.
5	Milwaukee Bucks* (6)	42.0	40.0	0.512	11	103.6	103.8	-0.
6	Indiana Pacers* (7)	42.0	40.0	0.512	11	105.1	105.3	-0.
7	Chicago Bulls* (8)	41.0	41.0	0.500	12	102.9	102.4	0.0
8	Miami Heat (9)	41.0	41.0	0.500	12	103.2	102.1	0.7
	Detroit							

Screen-scraping and automated interactions

- Login in
- Scrolling
- Pressing buttons

Sometimes easier than detective work in the javascript, but has RELIABILITY ISSUES.

Installing (and updating everytime something stops working) selenium

Selenium is an programmatic interface (api) to your browser.

You need to have firefox installed and a version of the Marionette (Download latest!! version here (https://github.com/mozilla/geckodriver/releases)) Then install selenium:

```
pip install selenium
```

or

```
conda install -c conda-forge selenium
```

If used on a server with no connection to a desktop screen: also install PyDisplay.

In [153]:
```
from selenium import webdriver
path2gecko = '/Users/axelengbergpallesen/Downloads/geckodriver' # define path to y
our geckodriver
browser = webdriver.Firefox(executable_path=path2gecko)
```

Login

In [169]:
```
#url = 'https://www.facebook.com'
#browser.get(url) # go to the page
#sel = '#email'# find css selector for the name field
#element = browser.find_element_by_css_selector(sel)# locate element
#name = 'robot_trespassing@ofir.dk'# define name
#element.send_keys(name)# send keys

# find password field
#sel = '#pass'
#element = browser.find_element_by_css_selector(sel)
# locate element
#password = 'thereyougo'
# define password
#element.send_keys(password)
# send keys

sel = '#loginbutton'
# Find button.
element = browser.find_element_by_css_selector(sel)
element.click()

# locate element
# click button
```

scrolling

some pages load more results when you reach the bottom of the page, here is a simple script that does this for you.

In [ ]:
```
browser.execute_script("window.scrollTo(0, document.body.scrollHeight);") # executing a simple javascript.
```

Advanced scraping methods

... ethics ... hacking

- detective work in the javascript.
- changing headers to access mobile content that might be more simple.
  - use https://www.whatsmybrowser.org (https://www.whatsmybrowser.org) from your phone to find the right headers you need.
- using proxies, ssh tunneling.
- Simulating human-like behaviour to avoid getting caught by anomaly detection methods.

In [ ]:

```
session = requests.session()
mobile_agent = 'Mozilla/5.0 (iPhone; CPU iPhone OS 7_1_2 like Mac OS X) AppleWebKi
t/537.51.2 (KHTML, like Gecko) Version/7.0 Mobile/11D257 Safari/9537.53'
session.headers['User-Agent'] = mobile_agent
session.proxies
```

APIs

- No parsing. Nice
- Fast and efficient.
- You can't get all that you can see.
- They change.
- Ratelimits

Through the API domains control what they want to share.

APIs can be for collecting data or it can be for accessing some service: e.g. tracking users of a website by directing traffic to google analytics, or getting geolocation data from the Google Geoencoding API (http://maps.googleapis.com/maps/api/geocode/json?language=da&address=1600 Pennsylvania Ave NW, Washington, DC 20500, USA).

Collecting data from APIs

Choosing an API: twitter, facebook, reddit, wikipedia, denmarks statistics, etc

Get to know the commands, rate limits, and errorcodes by experimenting or simply reading the documentation.

Today we will look at the Facebook API https://developers.facebook.com/docs (https://developers.facebook.com/docs)

instead of looking at the documentation we can go right to experimentations by using their graph explorer app https://developers.facebook.com/tools/explorer/ (https://developers.facebook.com/tools/explorer/).

Authentication

Basic authentication: Telling who you are, and a unique address so that can track you.

Specific permissions: Some queries is only allowed with special permissions.

```
* Application specific permissions, gathering user permission.
```

Building a query

In [196]:
```
# get an ID.
print(SoMe['Facebook'])
page_id = SoMe['Facebook'].split('/')[-1] # use the facebooklink collected earlier
```

```
https://www.facebook.com/100002941361500
```

In [187]:
```
# Authentication! set your access_token
base_url = 'https://graph.facebook.com/v2.10/%s'%page_id# define baseurl

fields = '?fields=feed.limit(10){message,id,to,from,comments,likes}'# define field
s
token = 'EAACEdEose0cBAHSnMGWyWm6gjS8mHzev0mUUzY8vILzFUiTZBITdUKYSFOAo4CarRUWi15lY
wVtHMZBVWLgKMOMTZCZAnkGaGpE3zzOcuITcSrvSO2v4Ohgsg1cuiWfJyHG1J4o28rixi1gAQZB1sn8rYS
okZAJ5xyv2sxY4Je5sQ6oFcK2catHG3XZC2SmM8kZD'
authentication = '&access_token=%s'%(token)
q = base_url+fields+authentication
print(q)
response = request(q)
```

```
https://graph.facebook.com/v2.10/185952168095456?fields=feed.limit(10){message
,id,to,from,comments,likes}&access_token=EAACEdEose0cBAHSnMGWyWm6gjS8mHzev0mUU
zY8vILzFUiTZBITdUKYSFOAo4CarRUWi15lYwVtHMZBVWLgKMOMTZCZAnkGaGpE3zzOcuITcSrvSO2
v4Ohgsg1cuiWfJyHG1J4o28rixi1gAQZB1sn8rYSokZAJ5xyv2sxY4Je5sQ6oFcK2catHG3XZC2SmM
8kZD
```

In [188]:

```
import json

d = response.json()
next_link = d['feed']['paging']['next']

def grab_next(response):
    try:
        link = response['feed']['paging']['next']
        return link
    except:
        return False
responses = []
while True:
    print('-')
    response = request(next_link).json()
    responses.append(response)
    next_link = grab_next(response)
    if not next_link:
        print('no more paging')
        break
```

```
('-',)
no more paging
```

In [168]:
```
## parsing json generic

## parsing json with the requests module
```

Rate limiting

APIs have rate limits, to ensure reliability you should make sure not break these.

Rate limits not always explicitly stated, need to test before launching program or create adaptable program.

In [ ]:

```
# waiting using the time module

# Intelligent rate limiting.
import time
logsize = 360
timestamps = [time.time()]*logsize
requestrate = 1 # number of calls pr second. If the ratelimiting error are thrown
update it.
ratelimiterrors = 0 #
def ratelimit():
    global timestamps
    global requestrate
    global ratelimiterrors
    servertime = time.time()
    timestamps.append(servertime)
    average_request_rate = (servertime-timestamps[0])/logsize
    if ratelimiterrors>0: # adopt new rate
        time.sleep(1800)
        requestrate+=0.01
        print 'updating requestrate to %r'%requestrate
        rateli miterrors = 0
    if average_request_rate < requestrate: # check if requestrate is to high
        time.sleep(1)
    timestamps.pop(0) # remove first timestamp
```

If you want reliable access you need to create an app. This will allow you to obtain access tokens that last for a month. Go to developers.facebook.com and create an app.

Converting the access token is done using the following code snippet:

```
url = 'https://graph.facebook.com/oauth
/access_token?client_id=%s&client_secret=%s&
grant_type=fb_exchange_token&fb_exchange_token=
%s'%(app_id,app_secret,access_token)
```

Make sure the access token is specific to you app.

Exercise 4

Collecting facebook data from the candidates.

First extract ids from the facebook links of each candidate

- beware of all the variations in the url.
- plus that some have names and not ids. If this is the case convert using the query = frank.jensen?fields=id,name
- some are pages and other are profiles, you can only get data from the pages.

In [ ]:

Reliability!

When using found data, you are the curator and you are responsible for enscribing trust in the datacompilation.

Reliability is ensured by an interative process, of inspection, error detection and error handling.

Build your scrape around making this process easy by:

- logging information about the collection (e.g. servertime, size of response to plot weird behavior, size of response over time, number of calls pr day, detection of holes in your data).
- Storing raw data (before parsing it) to be able to backtrack problems, without having to wait for the error to come up.